System-Level Programming Abstractions for Ubiquitous Computing

Robert Grimm, Janet Davis, Eric Lemar, Adam MacBeth, Steven Swanson,
Steven Gribble, Tom Anderson, Brian Bershad, Gaetano Borriello, David Wetherall

University of Washington
one@cs.washington.edu

Abstract

The promise of ubiquitous computing is predicated on the ability of programmers to develop applications that will be able to work properly in constantly changing situations. Functions that interact with users and infrastructure must be maintained at tolerable levels even if devices and users are roaming, users switch devices or engage new ones, and the network provides only limited services. In this position paper, we present an approach based on exposing programmers to change while providing them with tools that make dealing with highly dynamic systems much more straightforward. We are developing several applications using our infrastructure and have already released to the research community a version of our system, called *one.world*. Our next step is to build a user community and to this end we are preparing to offer a workshop to interested colleagues in the fall of 2001 where they will be able to prototype their own applications and leave with working code.

1 Introduction

The promise of ubiquitous computing is predicated on the ability of programmers to develop applications that will be able to work properly in constantly changing situations. Functions that interact with users and infrastructure must be maintained at tolerable levels even if devices and users are roaming, users switch devices or engage new ones, and the network provides only limited services [7]. Other factors may include changing bandwidth and latency of connections, power limitations of devices, and available interfaces to the user.

Our approach is based on the premise that we can't take the same road as much of the previous work in distributed computing and try to hide the distributed nature of the system from the application developer as much as possible. Transparency was appropriate when applications were distributed onto fairly uniform and full-featured computational resources and the connections between those resources were expected to be highly stable.

In ubiquitous computing, instead, it is clear to us that applications will demand to know how their operating environment is changing so that they can best adapt their computation, communication, and user interface. To take things even further, application elements may need to optimize power consumption and bandwidth on different nodes. We find it unlikely that we will be able to develop transparent solutions to these issues. It is far more likely that decisions will need to be application specific.

However, we recognize that exposing programmers to this level of dynamism will make their job much more difficult and error-prone. Today, much duplicated effort is expended in creating ad hoc solutions for dealing with these distributed systems problems rather than focusing on the actual applications. Therefore, our goal is to complement developers' awareness of change with tools that make dealing with highly dynamic systems much more straightforward and enhance reuse of code to deal with commonly encountered problems.

To make the developers' task feasible, we introduce a system architecture for ubiquitous computing, called *one.world*, that provides an integrated and comprehensive framework for building ubiquitous computing applications on large and dynamic computer networks. Our architecture does not introduce fundamentally new operating system technologies; rather the goal is to provide a set of services, such as check-pointing, migration, discovery, and replication that help to structure applications and directly simplify the task of coping with constant change.

In this position paper, we will only be able to provide a brief introduction to the key feature of our architecture. The remainder of the paper is structured as follows. In Section 2 we motivate some of the key abstractions that underpin

our system. Section 3 provides an overview of the services currently provided by our architecture and what we are planning for the immediate future. Section 4 describes two of the applications we are building with *one.world*, one of which is already being deployed to end users. In Section 5, we discuss our future plans which include a workshop for interested colleagues to be able to develop their own ubiquitous computing applications using *one.world*.

2 Abstractions

Three principal abstractions form the foundation of *one.world*. They capture two important aspects of ubiquitous computing applications, including: (1) the need for strong decoupling between code and data for purposes of composition and (2) the hierarchical encapsulation of code and data for purposes of migration.

First, tuples provide a common data format and simplify the sharing and querying of data. They are records with named fields and are self-describing in that an application can dynamically determine a tuple's fields and their types. They are used for storage, networking, and events.

Second, components implement functionality in applications and make it easy to compose units of functionality into larger entities. They interact by exchanging asynchronous events.

Finally, environments group stored data, application components, and hierarchically, other environments. They simplify management of users, their applications, and their data. They can be thought of as a combination of file system directories and nested processes in terms of more traditional operating systems [2, 3, 6]. They group related code and data so that they can be migrated together, for example.

These abstractions are crucial to the implementation of the services that will be used by developers of ubiquitous computing applications. They provide an ability to compose applications dynamically. We envision many ubiquitous computing applications that will share tuple-stores and possibly even some components. Decoupling of data and code will make it much easier to have applications interact through tuple-stores rather than tailoring applications to each other — there will be too many interacting to make this practical. Furthermore, we want to be able to compose applications at run time and have them evolve independently.

3 Services

Systems should expose change, including failures, rather than hide distribution, so that applications can implement their own strategies for handling changes. Leases [4] are an example of a suitable mechanism: they make time visible throughout a system and thus cleanly expose change.

At the same time, it is too inefficient to leave developers to their own devices for the development of each application. Our system architecture strives to provide considerable support for dealing with change in the form of primitives that are readily available to all applications developed on top of *one.world* [5]. Given these primitives, application developers can focus on making applications adaptable instead of creating necessary systems support. Thus, system support is provided in a reusable form through a set of services that form the core of our architecture. Here we list several of the most important: remote event passing, check-pointing, migration, discovery, and (soon) replication. These services are available to all applications developed on top of *one.world* and can be combined as needed.

Check-pointing captures the execution state of an environment tree and saves it as a tuple, thus making it possible to later revert to the environment tree's saved execution state. This is especially crucial when dealing with intermittent connections and devices that may be power-cycled unpredictably.

Migration provides the ability to move or copy an environment and its contents to another computation node either locally or across the network. It can affect any part or all of an application, because both components and stored tuples are moved or copied.

Discovery makes it possible to send events to services and applications whose location is not known. *one.world* discovery servers are automatically instantiated and provide event routing through both early binding, for efficiency, and late binding, for more dynamic situations.

Remote event passing (REP) provides the ability to send events to remote receivers, including receivers located by service discovery. This provides transparency as components move between nodes and itself makes transparent use of discovery services.

Finally, we will soon be providing *replication* to make stored tuples accessible on several nodes at the same time as well as duplicating application components. It provides a means for applications to decrease latency or handle

more interactions by utilizing multiple nodes. Hand-in-hand with replication we will be providing a framework for application- or component-specific reconciliation to resolve changes to tuple-stores that occur on multiple replicated copies (in addition to tagging and logging primitives necessary to keep track of all the changes).

4 Applications

We are developing several applications using the *one.world* architecture. These range from applications that deal with a single form of user interaction all the way to smart environments that are expected to continually add new user interfaces and devices. We briefly describe only two of them here.

one.radio allows users to have their working environment follow them from node to node. It currently supports the migration of instant messaging (so that a user always receives messages no matter where they are located) and audio (so that audio conferencing or radio streams also follow the user) but can be generalized to any *one.world* applications. Screen windows and application state migrate to another node as the user changes physical location. Users can pick up other applications (or, eventually, replicated copies of them) as they move about and have them join the collection that follows them around. Note that the node the applications began running on does not need to keep running. *one.radio* is currently under development with text messaging and radio applications already implemented.

Labscape is a smart-space for cellular biologists [1]. Its initial goal is experiment capture at enough detail to be able to reproduce experiments at a later date with different equipment. Eventually, it will be used for experiment automation via robots programmed by example. Currently, as biologists move about their office and laboratory spaces, they plan, research, and conduct several experiments simultaneously, their applications (in this case, experiment planning and indexing tools) move about the laboratory with them, collecting and organizing information about what they are doing within the context of their experiment. This integrates personal devices worn by the user with sensors attached to objects and instrumentation at different workbenches. Several versions of Labscape are deployed at UW's Cell Systems Initiative and being used by real biologists in their daily work. (See `http://labscape.cs.washington.edu`)

5 Closing remarks

In this paper, we have presented an approach to providing systems support for ubiquitous computing environments. These application platforms need to expose change, so that applications can implement their own strategies for adapting and systems, and they must support a high degree of decoupling, so that applications can be combined and extended at runtime.

We have introduced a system architecture for pervasive computing, called *one.world*, that achieves these goals. It cleanly separates data and functionality: tuples represent data and components implement functionality. Additionally, our architecture provides a set of powerful services, namely, check-pointing, migration, discovery, remote event passing, and replication, that serve as a foundation layer for many ubiquitous computing applications.

We have begun constructing applications using our framework and have already released to the research community a version of our system (available at `http://one.cs.washington.edu`). Currently, we are seeking to expand the number of both applications and developers using *one.world*. To this end, we are preparing to offer a workshop to interested colleagues in the fall of 2001 where they will be able to prototype their own applications.

References

[1] L. Arnstein, S. Sigurdsson, and B. Franza. Ubiquitous computing in the biology laboratory. *Journal of Lab Automation (JALA)*, 6(1), Mar. 2001.

[2] P. Brinch Hansen. The nucleus of a multiprogramming system. *Communications of the ACM*, 13(4):238–241, 250, Apr. 1970.

[3] B. Ford, M. Hibler, J. Lepreau, P. Tullmann, G. Back, and S. Clawson. Microkernels meet recursive virtual machines. In *Proceedings of the 2nd USENIX Symposium on Operating Systems Design and Implementation*, pages 137–151, Oct. 1996.

[4] C. G. Gray and D. R. Cheriton. Leases: An efficient fault-tolerant mechanism for file cache consistency. In *Proceedings of the 12th ACM Symposium on Operating Systems Principles*, pages 202–210, Dec. 1989.

[5] R. Grimm, J. Davis, B. Hendrickson, E. Lemar, A. MacBeth, S. Swanson, T. Anderson, B. Bershad, G. Borriello, S. Gribble, and D. Wetherall. Systems directions for pervasive computing. In *Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems*, pages 128–132, Elmau, Germany, May 2001.

[6] P. Tullmann and J. Lepreau. Nested Java processes: OS structure for mobile code. In *Proceedings of the 8th ACM SIGOPS European Workshop*, pages 111–117, Sept. 1998.

[7] M. Weiser. The computer for the twenty-first century. *Scientific American*, 265(3):94–104, Sept. 1991.